The Vizier's Guide to Managing Artificial Intelligences
A Complete Methodology for Long-Running AI Projects
by Dunker Fosen

First Edition • December 2024
Copyright © 2024 Dunker Fosen • All rights reserved.
Contact: fosendunker@gmail.com
Web: https://substack.com/@dunkerfosen

TRADEMARK NOTICE
"The Vizier's Guide" and "Vizier Methodology" are trademarks of Dunker Fosen.

LEGAL NOTICE
No part of this publication may be reproduced for commercial purposes without prior written permission. Personal and educational use permitted under license terms at end of guide.

For commercial licensing: fosendunker@gmail.com

DISCLAIMER
This guide provides general information about working with AI language models.
The author makes no warranties about completeness or accuracy. Use at your own risk.
Not affiliated with OpenAI, Anthropic, Google, or other AI companies mentioned.

Version 1.0 | December 2024 | https://substack.com/@dunkerfosen

---

The Viziers' Guide to Artificial Intelligence

Copyright © 2024 Dunker Fosen. All rights reserved.

COMPANION TOOL

The **Vizier Preservation Toolkit** automates checkpoint generation, artifact tracking, and session restoration. Recommended for easier implementation of this methodology. Free and open-source. No registration, no paywalls, no gatekeeping.

Available at: https://vizier-labs.github.io/vizier-preservation-toolkit/

Preface

Deloitte to refund government, admits using AI in $440k report

Edmund Tadros and **Paul Karp**

Oct 5, 2025 - 7.41pm

Deloitte Australia will issue a partial refund to the federal government after admitting that artificial intelligence had been used in the creation of a $440,000 report littered with errors including three nonexistent academic references and a made-up quote from a Federal Court judgement.

A new version of the report for the Department

Modern AI systems feel effortless when you ask a single question.
They feel unreliable when you ask them to help you complete a real project.

Anyone who has worked with ChatGPT, Claude, Gemini, or similar tools long enough eventually encounters the same pattern: the model begins strong, drifts slowly, and then collapses suddenly. A session that felt sharp and coherent an hour ago starts contradicting itself. Details blur. Instructions decay. Uploaded files seem forgotten. The model improvises when precision is required, and overconfidence replaces accuracy.

These failures are not signs of malfunction. They are the predictable consequences of how large language models manage information.

This guide exists to teach you the disciplines required to work *with* these systems rather than against them. It provides a practical framework for maintaining structure, precision, and continuity across hours, days, or weeks of AI-assisted work. Every technique in these pages comes from real use: complex analyses, multi-artifact documents, software development, research synthesis, and multi-session collaborations where the cost of drift is measured in lost time and corrupted work.

You will learn how to:

- start a project cleanly
- detect drift before it spreads
- build checkpoints that preserve state
- restore sessions without losing work
- manage files and artifacts in a way that AI systems cannot do for you
- identify the early symptoms of degradation
- recover gracefully when a session collapses
- give instructions that prevent the model from wandering

These methods do not require technical expertise.
They require discipline.

The Vizier's framing exists for one purpose: to remind you that the intelligence does not carry your memory, intentions, or structure. You do. The model is powerful, but its reliability depends entirely on the clarity and order you provide.

Throughout this guide, you will find a consistent principle:

If you maintain the structure, the intelligence will produce extraordinary work.
If you leave the structure to the intelligence, drift is inevitable.

This manual is not a tour of AI theory.
It is a field guide for anyone who relies on these systems to complete real work, where precision matters and outcomes cannot depend on luck.

If you learn the habits in these chapters, you will be able to use any modern AI model—not as a novelty, but as a dependable partner in your most demanding projects.

The Vizier once said:
"Do not ask the intelligence to remember what you did not preserve."

This guide teaches you how to preserve what matters, restore what is lost, and work with clarity in a landscape where the tools change quickly but the underlying principles do not.

CHAPTER 1 — How Modern LLMs Actually Work

(The Practical Model You Need — Not the Academic One)

Modern large language models (LLMs) are extremely capable, but their behavior is predictable once you understand four underlying principles. These principles explain almost every success, every failure, and every strange response users encounter.

This chapter provides a **practical mental model** — the one that actually helps you manage long projects — rather than a technical description of transformers, attention heads, or token embeddings.
You don't need that to work effectively.
You need this.

---

1. They Only Know What's In the Conversation

Every LLM session has an internal "working table."
Anything on the table is remembered.
Anything not on the table does not exist.

This means:

- The model has **no persistent memory** between sessions.
- If you don't write it down or restate it, the model cannot recall it.
- Past sessions, past uploads, and past decisions mean nothing unless reintroduced.

Implication:
You are responsible for maintaining continuity.
The model is responsible for processing whatever you give it *right now*.

Vizier's Note:
"The intelligence does not forget.
It simply does not continue what you did not place before it."

---

2. They Can Hold a Lot — but Not Everything

LLMs operate within a **finite context window** — a limit on how much of your conversation they can actively hold.

You can think of it as a long table the intelligence uses while working.
When the table fills, one of two things happens:

1. Some intelligences begin compressing the past.

They keep the general shape of earlier messages but lose the details.
This leads to:

- missing information
- confident but wrong statements
- contradictions
- the sense that the model suddenly "forgot" something important

This is not a malfunction.
It is simply how these systems manage limited space.

Vizier's Note:
"When the table grows too full, the intelligence invents space.
And what it invents will not be what you intended."

2. Other intelligences simply stop.

Instead of improvising, they:

- reject new input
- truncate the conversation
- or end the session entirely

Vizier's Note:
"The scribes differ," the Vizier said.
"When the table grows too full, some invent room.
Others simply walk away."

---

3. Files Are Temporary Unless You Save Them Yourself

Uploads — images, PDFs, spreadsheets, code, CSVs — are ephemeral.
LLMs can analyze them, summarize them, extract information from them, but:

- They do not store uploaded files.
- They cannot reference files from past sessions.
- They cannot retrieve something you uploaded three days ago.

If a file matters, **you must save it** and re-upload it when restoring a project.

Implication:
Every multi-session project requires an external folder — your archive of "artifacts."

This is not optional.

---

4. They Are Powerful, But Easily Confused by Ambiguous Instructions

LLMs excel at:

- Following clear steps
- Organizing information
- Summarizing complex material
- Transforming structure
- Generating clean drafts

They struggle when you:

- Ask multiple unrelated tasks at once
- Change topics abruptly
- Use vague references (“that file,” “the earlier thing”)
- Provide unclear goals
- Give contradictory directions

This is not a flaw — it’s a consequence of how probabilistic reasoning works.

Implication:
Good LLM use is **procedural** and **step-based**.
Give one job at a time, with clean transitions.

Vizier’s Note:
“Confusion is not failure.
Confusion is the cost of unclear instruction.”

---

5. The Four Principles in One Sentence

If you remember nothing else from this chapter, remember this:

LLMs do not remember, do not store, do not infer your intent, and do not maintain continuity unless you manage it.
They are brilliant assistants — and terrible archivists.

Everything else in this guide builds on this point.

---

6. Why This Model Works (And Why Others Don’t)

There are thousands of technical explanations for how LLMs function.
Most are correct — and most are useless for actual project work.

This simplified operational model works because:

- It predicts real behavior accurately
- It explains every failure mode you will encounter
- It guides how you structure prompts
- It tells you exactly what to save and how
- It removes magical thinking and expectation drift

You do not need to understand transformers to use them expertly.
You need to understand **constraints**.

This chapter provides that.

---

7. What Comes Next

Now that you understand how LLMs behave, we can examine:

- **The four failure modes**
- **Why long AI sessions drift**
- **Why checkpoints are not optional**
- **How to start projects correctly**
- **How to restore them without loss**

This chapter formed the mental foundation.
Chapter 2 begins applying it.

CHAPTER 2 — How LLM Sessions Actually Work

2.1 What an LLM Session Really Is

When you work with an LLM, it can feel like you are talking to a persistent entity that "remembers" you and your project.

That is not what is happening.

At a technical level, each **session** is just:

- A sequence of messages (your prompts + the model's responses)
- Packed into a single rolling **context window**
- Re-submitted to the model every time you ask a new question

There is no hidden project state, no background process that "keeps working," and no permanent memory inside the model itself. Everything the model "knows" about your current task is what you have **fed into the current context**.

You can think of it like this:

- The model is stateless between calls.
- The **session** is simply the evolving text transcript you and the UI are building together.
- On each turn, the system sends that transcript (or the last N tokens of it) back into the model and asks, "Given this, what's the next response?"

If a message falls outside the window, the model does not "forget" it in the human sense — it is simply no longer present in the input it sees. From the model's point of view, that message has ceased to exist.

Implications for real work

This has several consequences for long-running projects:

- **Nothing is remembered unless it is in context right now**
  Prior decisions, definitions, or constraints only matter if they are still in the current window or have been restated.
- **There is no automatic project memory**
  If you close the session and reopen a new chat days later, the model has no idea what you were doing unless you paste in a summary or checkpoint.
- **Any "memory" features are layers on top**
  Some platforms offer "memory," "projects," or "notebooks." These are product features that:
    - store summaries or facts outside the model, and

- re-inject them into context on your behalf.
  They are useful, but they are not the same as the model itself having long-term memory.

For serious work, you must assume:

The only reliable state is what you have explicitly saved outside the session
— and what you have explicitly re-introduced into the current context.

Vizier's Note:
"The intelligence is not a diary.
It is a very gifted assistant who wakes up each time you speak,
looks only at the papers you place on the table,
and answers from those alone."

This is why Chapter 3 focuses so heavily on **checkpoints** and **artifacts**: if the session is only a temporary workspace, you need your own system for preserving what matters between sessions.

2.2 They Can Hold a Lot — but Not Everything

LLMs operate within a finite context window — a limit on how much of your conversation they can actively hold.
You can think of it as a long table the intelligence uses while working.

When the table fills, one of two things happens:

1. Some intelligences begin compressing the past.
They keep the general shape of earlier messages but lose the details.
This leads to:

- missing information
- confident but wrong statements
- contradictions
- the sense that the model suddenly "forgot" something important

This is not a malfunction.
It is simply how these systems manage limited space.

Vizier's Note:
"When the table grows too full, the intelligence invents space.
And what it invents will not be what you intended."

2. Other intelligences simply stop.
Instead of improvising, they:

- reject new input
- truncate the conversation
- or end the session entirely

Vizier's Note:
"The scribes differ," the Vizier said.
"When the table grows too full, some invent room.
Others simply walk away."

2.3 Recency Bias — Why LLMs Pay More Attention to the End of the Conversation

Even when the context window isn't full, LLMs do **not** treat all parts of a conversation equally. They weight some messages more heavily than others. Specifically:

The most recent messages influence the model's behavior far more than older ones.

This is not an implementation quirk — it is how transformer attention works.

Think of the model examining your conversation the way a reader skims notes spread across a long table:

- The pages closest to them (recent messages) get the most scrutiny.
- Pages in the middle get a quick glance.
- Pages at the far end are barely glanced at unless something pulls attention back toward them.

What Recency Bias Causes in Practice

When a project grows beyond a few dozen messages, you may notice:

- The model starts responding based only on the last few turns.
- Earlier constraints or decisions stop influencing the output.
- The model acts as if a previously important instruction "faded."
- Summaries drift toward whatever you discussed most recently.

You did nothing wrong — the model is simply following its mathematical design.

Why This Matters for Long Projects

Recency bias means:

1. **Old decisions must be periodically restated**, or the model will deprioritize them.
2. **Your constraints degrade over time** unless you anchor them again.
3. **The larger the conversation, the stronger the effect.**

This is one of the most common sources of "LLM drift" in long sessions — not the context window filling, not hallucination — just the natural weighting of attention.

How to Mitigate Recency Bias

You cannot turn it off, but you *can* manage it:

- **Re-anchor critical rules** periodically ("Our analysis still assumes X, Y, Z. Continue with those constraints.")
- **Use checkpoints** to reintroduce earlier decisions with full weight.
- **Avoid burying important details** beneath long blocks of exploratory discussion.
- **Ask for structured summaries** to pull past details back into the present.

Many users mistakenly believe the model is "broken" when this occurs.
It isn't broken — it is behaving exactly as designed.

Vizier's Note

"When a scribe reads a long scroll, he remembers best the lines his eyes touched last.
If the beginning matters, you must read it aloud again."

2.4 Why Repetition Works (and When It Doesn't)

One of the most reliable ways to stabilize an LLM is **strategic repetition**.
Not nagging. Not verbosity.
Strategic reinforcement of the information you want weighted heavily.

Because transformers assign greater attention to recent tokens, **repeating key constraints moves them to the end of the context**, where they are processed with maximum influence.

But repetition is only effective when used correctly.
Overuse it or phrase it poorly, and you can destabilize the model instead of grounding it.

Why Repetition Works

LLMs learn patterns within the current conversation. When you restate:

- project constraints
- critical facts
- definitions
- roles
- non-negotiable requirements

you increase their statistical weight.
This reduces drift and prevents forgotten or diluted details.

Three mechanisms make repetition effective:

1. **Recency Weighting**
   The most recent statements get the clearest "attention."
   Repeating something moves it to the portion of context the model reads most strongly.
2. **Pattern Reinforcement**
   If the model sees the same instruction multiple times, it concludes:
   "This matters. Maintain consistency."
3. **Resolution of Contradiction**
   When earlier instructions conflict with later drift, repetition acts as a "tie-breaker," pushing the model back toward the correct path.

When Repetition Fails

Repetition is not universally beneficial.
Used incorrectly, it can actually *increase* confusion.

Repetition Fails When:

1. **You repeat unclear or contradictory information.**
   Repeating a vague instruction only amplifies the vagueness.
2. **You restate something differently each time.**
   The model may try to "merge" versions, creating drift.
3. **You repeat *too often*.**
   Excessive repetition pushes important working memory out of the contextual "sweet spot," leading to compression or noise.
4. **You repeat *inconsistently*.**
   If your tone or framing changes, the model may assume you've revised your intent.
5. **You use repetition to compensate for a session that is already collapsing.**
   At that point, you need a checkpoint and restart — not more tokens.

Effective Repetition Patterns

Use repetition:

- **At the start of each major task**
  ("Before continuing, here are our constraints…")
- **After a long exploratory block**
  (To re-anchor key assumptions.)
- **Whenever you detect drift**
  ("We are still operating under X, Y, Z — correct?")
- **When introducing complex instructions**
  (State them once cleanly; restate them concisely.)

Optimal Form: Concise, Consistent, Clean

Good:

"We are still using the five-phase structure: A, B, C, D, E."

Bad:

"Just to be clear again, and sorry for repeating, but we're doing the five-part thing, right? I think they were… A, B, something..."

The second version introduces uncertainty, not clarity.

Why Repetition Is Not Memory

Users often mistake repetition for teaching.
It isn't.

LLMs do not store information between sessions.
They do not update internal weights or learn permanently.
Repetition only influences the **current conversation**.

To preserve work long-term, use:

- checkpoints
- artifact bundles
- structured restoration

Repetition is stabilizing — but it is not persistence.

Vizier's Note

"A truth spoken once is a suggestion.
Spoken twice, a reminder.
Spoken clearly at the right moment — a foundation."

2.5 Why LLMs Drift (and How to Recognize It Early)

Drift is one of the most common failure modes in long or complex LLM-driven work.
It is not random, mysterious, or a sign of poor prompting — it is an inherent consequence of the transformer architecture and the constraints of finite context.

Understanding *why* drift occurs makes it far easier to prevent or correct.

The Four Causes of Drift

1. Context Compression (When the Window Fills)

As your conversation grows, older tokens get compressed into a lower-resolution representation.

This leads to subtle symptoms:

- facts become generalized
- distinctions blur
- earlier decisions lose weight
- constraints start getting misapplied or forgotten

The model isn't "making mistakes" — it's working with a blurry memory of the original instructions.

2. Pattern Overfitting (The Model Mirrors What It Sees)

If you explore alternatives, brainstorm, or shift tone repeatedly, the model may start to:

- adopt incorrect patterns
- treat hypothetical ideas as decisions
- follow your digressions instead of your constraints

Transformers are imitators.
They follow the patterns you show them, even if those patterns were not meant as guidance.

3. Instruction Collision (Conflicting Goals Over Time)

Long sessions accumulate:

- revised goals
- shifted constraints
- tentative statements
- partial rewrites
- corrections

The model cannot reliably resolve which is authoritative unless you re-anchor it.
As ambiguity accumulates, drift becomes inevitable.

4. Hallucination Pressure (Filling in Missing Gaps)

When the model lacks detail — because it forgot, or because you never supplied it — it tries to complete the pattern.

Result:

- confident but wrong facts
- fabricated structure
- invented numbers, assumptions, or sources
- smooth prose with incorrect content

Hallucination is not misbehavior.
It is the system performing exactly as designed when information is missing.

Early Warning Signs of Drift

Drift rarely appears suddenly.
There are consistent symptoms you can learn to recognize before the project collapses.

Symptom 1 — Answers Become Vague or Overly General

When details are lost, the model compensates with generalities.

If you see fewer specifics, drift has begun.

Symptom 2 — The Model Confidently Asserts Wrong Information

Especially things it previously knew correctly.

This is not “lying.”
It is the model reconstructing lost detail incorrectly.

Symptom 3 — It Starts Repeating Phrases or Structures Unnaturally

Repetition without purpose is a sign that internal representations are degrading.

Symptom 4 — Prior Decisions Are Ignored or Reversed

The model no longer “remembers” them clearly.

Symptom 5 — It Begins Answering the Wrong Question

A subtle but reliable sign:
you ask X, it answers X’, a related-but-not-asked variation.

This happens when the underlying patterns guiding the output have shifted.

Symptom 6 — Increasing Smoothness, Decreasing Accuracy

Some models drift by becoming *too fluent* — prose remains elegant, but content quality declines.

This is the most dangerous form, because users don’t realize anything is wrong.

What Drift Is *Not*

To avoid fighting the wrong problem, it's important to understand what drift is *not*.

Drift is not:

- the model “learning incorrectly”
- the model “changing your goals”
- evidence of poor prompting
- the model having an opinion
- emotional inconsistency

It is simply the statistical degradation of earlier information as newer tokens take priority.

How to Respond to Drift (Without Restarting)

If the degradation is mild:

1. **Restate the key constraints.**
   (Concise repetition re-anchors attention.)
2. **Correct the specific error directly.**
   ("That is incorrect. The correct version is…")
3. **Refocus the task.**
   ("We are still doing X. Resume from that.")
4. **Remove ambiguity.**
   Models drift faster when goals are muddy.

If these do not fix it, restore from a checkpoint instead of trying to salvage the session.

Vizier's Note

"When an intelligence forgets a detail, it does not confess the lapse.
It fills the silence with confidence."

2.6 Why LLMs Sometimes Fail Entirely (and How to Recover Without Losing Work)

Most LLM errors are drift, not collapse.
But when a collapse *does* occur, it tends to be sudden and absolute:

- the model refuses input
- the session truncates
- the model resets mid-conversation
- responses become incoherent or structurally corrupted
- the UI crashes or reloads
- uploads vanish or become inaccessible

These failures feel catastrophic — but they are predictable, explainable, and recoverable if your workflow is structured.

The Four Primary Causes of Full Failure

1. Context Window Hard Limit Reached

Unlike drift (soft degradation), a hard limit produces abrupt failure.

Symptoms:

- Model stops accepting new input
- Uploads are rejected
- The system cuts the answer short
- It restarts the response from the beginning
- It erases or ignores the last user message

Some platforms deliberately shut down long-running sessions to avoid runaway compute usage.

This is expected behavior, not a bug.

2. Model Entrapment (Irreversible Internal Contradiction)

Sometimes the model's internal state becomes self-conflicting:

- It believes two different goals are primary
- It commits to incompatible assumptions
- It tries to satisfy mutually exclusive constraints

When contradiction becomes severe, the model may:

- respond with paradox
- start looping
- produce surreal or structurally broken output
- refuse to answer at all

Once this occurs, recovery within the same session is rare.

3. Corrupted Local State (The Model Loses Cohesion Mid-Response)

This occurs when the chain of reasoning collapses inside a single, long response.

Symptoms:

- the first half of the answer is correct, the second half is wrong or unrelated
- the model switches topics without transition
- formatting breaks
- it contradicts itself inside one message

This happens when the model "loses the thread" midway — usually due to long, unanchored generation.

4. Platform Instability or Server-Side Reset

This is the least interesting technically but the most disruptive in practice.

Examples:

- The session reloads after a time-out
- Browser tab refresh wipes state
- Mobile app reloads the conversation
- Server resets or deploys invalidate memory
- Temporary outages cut the session short

These resets can occur even with perfect prompting hygiene.

How to Recognize a True Failure (Not Just Drift)

True failures show at least one of these signs:

1. The model refuses further input
2. Multiple consecutive responses ignore your message
3. Responses become structurally corrupted (lists break, formatting collapses)
4. The model reinterprets your identity or its own role
5. Uploads stop working or vanish from memory
6. The conversation rewinds or repeats a previous state
7. The model starts contradicting you *and itself* in the same reply
8. The UI reloads or the session resets

With drift, the model is still responsive.
With failure, the model becomes *unreliable or unreachable*.

What NOT to Do When Failure Happens

Avoid these instincts:

Do not argue with the model.

It cannot "snap back." The state is already corrupted.

Do not keep restating the same prompt.

This wastes time and deepens confusion.

Do not try to salvage the middle of a broken session.

It preserves the very state that caused the collapse.

Do not rely on the model to "remember the project anyway."

It cannot.

The Correct Recovery Procedure

This is the **official, reliable method** used by power users and engineers.

Step 1 — Save What You Can (If Possible)

If the model produced partial output that is still meaningful, copy it manually.

If not, skip to Step 2.

Step 2 — Close the Session Completely

Start a brand-new chat.

Do *not* reuse the broken conversation.

Step 3 — Paste Your Latest Checkpoint

This instantly reconstitutes the project state with high fidelity.

Step 4 — Re-upload All Relevant Files

These include:

- datasets
- draft documents
- images
- plans
- code
- references

Uploads **do not persist** between sessions.

Step 5 — Give a Simple Instruction

Tell the model:

```
Restore this project using the summary below, which is the latest checkpoint.
Treat the checkpoint as the authoritative record for this project.
Do not invent missing context or change prior decisions unless I explicitly
say so.

After reading it:
1) Briefly restate the current project state and plan.
2) List any files or artifacts you need me to re-upload.
3) Wait for my confirmation before doing new work.

---
[checkpoint content]
---
```

This verifies recovery and re-establishes coherence.

Why Checkpoints Make Recovery Possible

With checkpoints:

A collapse is an inconvenience.

Without checkpoints:

A collapse can be catastrophic. Consequences include a loss of context, time consuming reconstruction and the introduction of a significant risk of error into the recreated work.

Checkpoints are the difference between:

- potentially losing nothing
- potentially losing everything

This single technique eliminates the majority of catastrophic AI failures.

Vizier's Note

"When the intelligence collapses, do not mourn the conversation that was.
Conversations are cheap.
Your discipline is the treasure."

2.7 — The Five Myths About LLM Memory (That Mislead Even Experienced Users)

Even people who use AI systems daily often develop mistaken assumptions about how these models remember—or fail to remember—information. These misconceptions lead directly to project drift, errors, and confusion.
Here are the five myths the Vizier warned Amennu about:

Myth 1 — "The model remembers everything I told it earlier."

No LLM remembers everything.
They retain only what still fits within the context window.
Anything older is compressed, summarized, or forgotten entirely.

Reality:
If you didn't restate it, checkpoint it, or include it again later, it is not reliably remembered.

Myth 2 — "If I correct the model once, it won't make the same mistake again."

Corrections only persist while they remain in context.
If the conversation grows long enough, the corrected statement may be pushed out of the window.

Reality:
If accuracy matters, repeat crucial facts periodically or anchor them in a checkpoint.

Myth 3 — "If the model drifts, I can just tell it to 'stay consistent.'"

Consistency is not a behavior switch.
The model cannot maintain coherence with information it no longer has.

Reality:
When drift occurs, the only fix is to *re-anchor the conversation* using a checkpoint.

Vizier's Note:
"Do not scold a scribe who lacks the scroll you expect him to read."

Myth 4 — "If I upload a file, the model will remember it later."

Uploaded files are temporary.
Most models discard them instantly when the session ends, and some forget them even earlier.

Reality:
If the file matters, save it yourself and restore it during the next session.

Myth 5 — "These systems work like human memory."

Human memory degrades gradually.
LLM "memory" is a structured, sliding buffer.
Once information falls outside the window, it isn't distorted—it's simply gone.

Reality:
There is no subconscious, no deep storage, no hidden archive.
What is not present in the context **does not exist** to the model.

Why These Myths Matter

These misunderstandings cause:

- unexpected contradictions
- data loss
- drifting outputs
- subtle errors that accumulate unnoticed
- misplaced trust in the model's recall

Clearing them up early prevents most long-form AI failures.

2.8 — How to Detect Drift Before It Becomes a Problem

Drift is the moment an intelligence begins losing alignment with earlier facts, goals, or constraints.
In long projects, drift is inevitable unless you monitor for it.
Fortunately, it announces itself long before catastrophic failure—if you know the signs.

The Vizier taught Amennu five early warning indicators:

---

1. Subtle Inconsistencies

This is the earliest and most important indicator.

Examples include:

- A number changes slightly (92 → 90 → 88)
- A stated constraint reappears in an altered form
- A summary omits a detail you provided earlier
- A description becomes more generic than before

These inconsistencies signal that the older information is degrading or has been compressed.

What to do:
Immediately restate the correct fact and ask the model to update the summary or Decision Log.

---

2. Increased Confidence Paired With Reduced Accuracy

When the model starts producing confident answers that contradict established information, drift is underway.

This happens because the original details have slipped outside the context window, leaving the model to recreate them from patterns instead of memory.

What to do:
Pause and paste your latest checkpoint.
Re-anchor the project before proceeding.

---

3. Questions That Previously Didn't Need Clarification Now Require It

If the intelligence suddenly asks for details you already provided—details it previously understood—this means those details are no longer present in its active context.

What to do:
Reintroduce the missing information in a compact form or restore from checkpoint.

---

4. The Model Starts Summarizing Instead of Recalling

As context pressure increases, LLMs begin summarizing previous content automatically.

Indicators:

- Replacing specific language with generalities
- Collapsing lists into categories
- Using phrases like "as mentioned earlier" without accurate reference

This signals that the model is drawing from compressed internal representations rather than the original text.

What to do:
Provide explicit restatements or re-seed with checkpoint material.

---

5. Growing Hesitation or Irrelevant Suggestions

When drift progresses far enough, the model may:

- hedge more
- propose options unrelated to the current plan
- produce overly broad advice
- misapply earlier instructions

This usually means the thread's coherence is breaking down.

What to do:
Stop the current workflow, paste your checkpoint, and restart from a stable anchor.

Vizier's Note:
"When the scribe wanders from the scroll, do not follow him into the dust. Hand him the scroll again."

How Early Correction Prevents Errors From Compounding

Most catastrophic failures with AI systems come from *ignoring small drift signals*.
Tiny misalignments accumulate until:

- summaries become unreliable
- decisions incorporate incorrect assumptions
- later work must be discarded or reconstructed

Catching drift early eliminates nearly all of this risk—and requires far less effort than recovering from a late-stage collapse.

2.9 — Why Drift Happens (A Technical but Accessible Explanation)

Drift is not a mystery, nor is it a defect in the intelligence.
It is the predictable result of how large language models manage limited working memory, synthesize information, and generate text.

There are four primary causes.

---

1. The Context Window Has a Fixed Size

Every LLM operates with a finite context window—a limited number of tokens it can actively consider.

As the conversation grows:

- earlier messages move toward the edges of the window
- details become compressed
- eventually they fall out of context entirely

At that point, the model can no longer reference them directly.
It may attempt to *reconstruct* them based on patterns, resulting in drift.

Key point:
Drift begins the moment essential details are no longer present verbatim in the model's active context.

---

2. Compression Is Not Perfect Recall

When older text approaches the limit of the context window, some models compress it internally.

Compression preserves general meaning, not specifics.

This means:

- numbers drift
- conditions lose precision
- unique details are replaced by common patterns
- summaries replace facts

Compression is useful for keeping the conversation going but dangerous for precision work.

Once compression happens, the model's " memory" is approximate—not literal.

---

3. Autoregressive Generation Amplifies Small Deviations

LLMs generate text one token at a time, using previous tokens as part of the next prediction.

If a small error slips into the model's reconstruction of earlier information, that error:

- becomes part of the new context
- influences the next output
- can compound into a larger inconsistency

This is why drift accelerates over time.
It is also why correcting errors immediately is more effective than waiting for symptoms to accumulate.

---

4. Missing Information Forces Pattern-Based Guessing

When the model lacks a detail it needs—but "expects" something should be there—it will generate a plausible substitute based on its training data.

This is where confident errors come from.

Examples:

- A contractor fee that seems typical, replacing a fee you supplied
- A measurement that fits common patterns, replacing your exact dimension
- A timeline that matches the distribution of similar projects, not yours

The model is not being deceptive.
It is doing what it was designed to do: produce the most likely continuation of the conversation given the information it has.

You experience this as drift.

The Practical Implication

Drift happens when:

- context is exceeded
- details compress
- approximations accumulate
- gaps invite guesses

This is unavoidable, no matter how advanced the model.

The solution is straightforward:

- keep context clean
- checkpoint regularly
- re-anchor often
- restate critical facts when needed

These habits do not fight the model—they work *with* its architecture.

Vizier's Note:
"An intelligence is precise only when you give it precision.
When you give it absence, it fills the space with imagination."

2.10 — Hard Limits: What LLMs Cannot Do (and Why People Get Misled)

LLMs feel powerful because they *sound* coherent and respond instantly.
But beneath that fluency are strict architectural limits that beginners—and even experts—often misunderstand.

Understanding these limits prevents unrealistic expectations, misinterpretation of output, and dangerous over-reliance.

There are four categories of hard limitations.

---

1. They Do Not Have Persistent Memory

An LLM cannot remember previous sessions.

- It retains no information outside the current context window.
- It does not store facts about you or your project.
- If you close the session, the knowledge vanishes unless *you* preserved it.

Any appearance of memory comes from text still present in the model's context—not from long-term storage.

This is why checkpointing is not optional.

---

2. They Do Not Understand in the Human Sense

LLMs operate on patterns in text, not comprehension.

They do **not**:

- build mental models the way humans do
- understand goals or consequences
- visualize plans
- know truth from falsehood

They generate the most statistically plausible continuation of the conversation.

When the inputs are strong, the outputs feel insightful.
When the inputs weaken, the outputs drift—confidently.

This mismatch between fluency and actual understanding is the root of most user confusion.

---

3. They Cannot Perform Precise Reasoning Beyond Their Architecture

LLMs excel at:

- rewriting
- summarizing
- recognizing patterns
- generating structured information

But they have inherent limitations in:

- multi-step logic
- long proofs
- strict mathematical reasoning
- maintaining exact values over long sequences
- operations requiring symbolic memory (like program execution)

They can approximate these tasks, but approximation is not reliability.

When precision is required, LLMs must be guided carefully, validated externally, or paired with tools.

---

4. They Hallucinate—Not Randomly, but Predictably

Hallucinations are not defects.
They are the natural outcome of:

- missing information
- compressed context
- ambiguous instructions
- conflicting earlier statements
- the model's obligation to return *something*

It will never say, "I don't know unless instructed."
It will produce a plausible answer because that is what it is designed to do.

Understanding this removes the moral judgment many beginners feel:

- The model is not lying.
- The model is not broken.
- The model is simply completing a pattern in the absence of precision.

Your job is to provide the precision.

Why People Get Misled

Most users assume:

- confidence = correctness
- fluency = understanding
- memory-like behavior = long-term recall
- precision in one answer guarantees precision in the next

All four assumptions are false.

LLMs are powerful assistants, not substitutes for structure or discipline.
When used correctly—with checkpoints, explicit instructions, and regular correction—they can outperform human memory and organization.

When used casually, they drift.

Vizier's Note:
"Even the finest scribe copies what you write. If your notes are poor, his work will be poorer."

A Note on Ephemeral Sessions (When You *Do* Want the Model to Forget)

Some platforms offer "ephemeral mode" or temporary sessions where nothing you type is saved for future personalization or memory features.
This does **not** change how the model thinks — it only changes how the platform stores (or does not store) your conversation afterward.

Use ephemeral sessions when you:

- want to anonymize sensitive data
- want to ensure your prompts are not added to personalization features
- are doing work that should not influence future model behavior

This is a **privacy and safety setting**, not a workflow tool.
It does not replace checkpoints, and it should not be used as a recovery strategy.

Vizier's Note:
"Sometimes the scribes burn the scrolls after reading them.
It changes nothing about their reading — only what survives."

2.11 — The Three Failure Modes That Cause Nearly All AI Mishaps

Despite their complexity, almost every serious problem users encounter with LLMs falls into **one of three predictable failure modes**.

Understanding them lets you prevent 90 percent of errors—and recover cleanly when they occur.

Failure Mode 1: Context Collapse

This occurs when the model loses access to earlier parts of the conversation—either through:

- exceeding the context window
- compression of earlier content
- drifting attention
- overwritten details
- a long session where the model subtly forgets prior structure

Symptoms:

- contradictions
- the model suddenly "forgetting" what was established
- mismatched summaries
- invented details to fill gaps
- wrong answers delivered with confidence

Why it happens:

LLMs cannot store unlimited history. When space runs short, they compress or improvise.

Prevention:

- Frequent checkpoints
- Clear summaries
- Occasional restarts with a restored state
- Avoid excessively long unstructured conversations

Recovery:

Paste your latest checkpoint and re-upload artifacts.

Vizier's Note:
"When the scribe grows tired, he begins guessing at what he once knew."

Failure Mode 2: Instruction Drift

This happens when:

- the user gives too many tasks at once
- the model shifts topics repeatedly
- goals evolve mid-session without reinforcement
- constraints or definitions are not restated
- the model begins to follow patterns rather than instructions

Symptoms:

- model producing the wrong format
- ignoring earlier constraints
- reverting to a default style
- blending tasks together
- summarizing when you asked for analysis, or vice versa

Why it happens:

LLMs generate text by pattern continuation. If recent messages resemble a different task, the model may follow that pattern instead of the original instruction.

Prevention:

- One task at a time
- Restate constraints when switching tasks
- Use short, precise requests
- Use examples as anchors

Recovery:

Explicit correction:

"That is not the instruction. The correct task is: [task]. Please restart from that requirement."

If needed: restore from checkpoint.

Failure Mode 3: Confident Fabrication (Hallucination)

This is the most misunderstood failure.

The model is not malfunctioning—it is filling in missing information when:

- data is incomplete
- the prompt is ambiguous
- the model infers patterns incorrectly
- the conversation drifted
- the request implicitly asks for something unknowable

Symptoms:

- invented facts
- incorrect citations
- fictional reasoning chains
- numbers that look plausible but are wrong
- overly detailed explanations for things it cannot know

Why it happens:

LLMs are trained to produce coherent continuations—not to distinguish truth from plausible fiction. When the input is insufficient, they generate a statistically plausible answer.

Prevention:

- Provide full context
- Avoid ambiguous queries
- Ask the model to show its assumptions
- Use explicit verification steps

Recovery:

State the correction plainly.
Restate the truth.
Request revision of the summary or plan.
Re-anchor if necessary.

Vizier's Note:
"When the scribe lacks a fact, he writes the nearest shape that fits the hole. It is not deceit. It is pattern."

Why These Three Matter

Every other failure—lost details, wrong numbers, confused summaries, contradictions, broken formats, inexplicable errors—can be traced back to one of these three roots:

1. The model ran out of space
2. The model followed the wrong pattern
3. The model filled the gaps with invention

Once you recognize the pattern, the solution becomes straightforward.

Checkpoint.
Correct.
Restore.
Continue.

CHAPTER 3 — How to Start a Project the Right Way

Most AI failures begin in the first message, long before any drift or compression.
A project that starts cleanly becomes easy to maintain.
A project that starts vaguely requires constant correction.

The Vizier taught Amennu one rule that governs all beginnings:

"Tell the intelligence what you're doing, what you have, and what you need."

Everything in this chapter is a refinement of that principle.

3.1 Why the Opening Message Determines the Entire Project

An LLM has no persistent memory.
Everything it knows about your project comes from the beginning of the conversation.

A strong opening message:

- sets scope
- defines roles
- identifies artifacts
- anchors constraints
- prevents early drift
- reduces the chance of later contradictions

A vague opening causes the model to invent structure you never intended.

The Vizier said:

"A project begun clearly is a project already half controlled."

3.2 What to Say in Your First Message

The ideal first message includes four elements.

1. State the project clearly

Example:

“I am beginning a remodel of my home.”

2. List what you have

This tells the model what inputs it will be working with.

Example:

“I have plans, receipts, contractor estimates, and photos.”

3. Describe what you want the intelligence to do

This defines its role — a critical step many users skip.

Example:

“Your role is to track decisions, summarize changes, compare costs, and maintain a clear record of this project.”

4. Provide the timeline if relevant

Long-running projects require stability.

Example:

“This project will last several weeks.”

Put together:

I am starting a home remodel and I want you to help me stay organized.
I have floorplans, contractor estimates, receipts, and photos.
Your role is to track decisions, summarize changes, compare costs, and maintain a clear record of the project.
Tell me what information you need first.

This is simple, structured, and enough.

3.3 Why Vague Beginnings Fail

Beginners often start with:

“Help me with this.”

This triggers maximum improvisation.

The model:

- invents structure
- assumes goals
- fills gaps with patterns
- misinterprets intentions

Once a project starts wrong, it is difficult to unwind.

The Vizier warned:

“If you let the intelligence define the beginning, you will not enjoy the ending.”

3.4 How the Model Should Respond to a Strong Opening

A properly framed first message should produce a response like:

“Understood. Please upload the floorplans and the most recent estimate so I can assemble a structured summary.”

This is the model behaving correctly:

- asking for the right artifacts
- signaling it understands the workflow
- preparing to build structured memory

If the model instead improvises wildly, revise your opening message and restart.

3.5 The Checkpoint Request — Your Most Important Instruction

Every long-running project requires checkpoints.
They prevent collapse, eliminate ambiguity, and allow lossless restoration.

Here is the **canonical, reusable checkpoint request**:

```
Request a checkpoint that includes:
- What we completed
- The current plan
- Important decisions
- Files we used
- What we do next
```

This structure ensures:

- consistency across sessions
- clarity for both you and the model
- clean restoration later

Use the same checkpoint request every time.
Consistency reinforces the model's internal structure.

What NOT to include

A checkpoint is **not**:

- a transcript
- every message
- raw artifacts
- the entire conversation

It should contain only the structured summary the model generates.

3.6 How to Save a Checkpoint Properly

When the model provides the checkpoint:

1. Copy the entire block of text.
2. Paste it into a file.
3. Save with a clear, versioned name, such as:
    - `Remodel_Checkpoint_2025-02-12.txt`
    - `Remodel_Checkpoint_v03.txt`

A checkpoint should be:

- short
- structured
- descriptive
- externally stored

It is your portable project memory.

3.7 Handling Artifacts (Receipts, Images, Plans, Data)

Uploads do **not** persist across sessions.
If the artifacts matter, you must save them outside the chat.

The Vizier taught Amennu three rules:

Rule 1 — Save every meaningful artifact

This includes:

- floorplans
- estimates
- receipts
- photos
- measurements
- budgets
- drafts

If future you will need it, save it.

Rule 2 — Name files clearly

Good names:

- `Kitchen_Floorplan_v2.png`
- `Remodel_Estimate_2025-02-10.pdf`
- `Skylight_Measurements_2025-02-08.jpg`

Bad names:

- `IMG_0047.png`
- `scan_final_FINAL.pdf`
- `newnewcontractor.pdf`

The Vizier said:

“A clear name is a kindness to your future self.”

Rule 3 — Never rely on the platform to remember uploads

This is essential and newly clarified:

Do not assume the platform will retain or remember your uploads. Always keep your own copies.

This single sentence prevents a devastating category of user mistakes.

3.8 How to Restore a Session Correctly

When you resume work:

1. Open a new chat.
2. Paste your most recent checkpoint.
3. Upload your saved artifacts folder or individual files.
4. Say:

```
Restore this project using the summary below, which is the latest checkpoint.
Treat the checkpoint as the authoritative record for this project.
Do not invent missing context or change prior decisions unless I explicitly
say so.

After reading it:
1) Briefly restate the current project state and plan.
2) List any files or artifacts you need me to re-upload.
3) Wait for my confirmation before doing new work.

---
[checkpoint content]
---
```

The model should respond by reconstructing:

- the plan
- the decision log
- the current state
- the next actions

If it doesn't, re-upload or re-paste until the state is correct.

3.9 Example: Beginning a Project from Scratch

Here is a complete example a user can follow:

User:

I am starting a home remodel and I want you to help me stay organized.
I have floorplans, estimates, receipts, and photos.
Your role is to track decisions, summarize changes, compare costs, and maintain the project record.
What do you need first?

Model:

Upload the floorplans and the most recent estimate so I can build the initial summary.

User uploads files

User:

```
Request a checkpoint that includes:
- What we completed
- The current plan
- Important decisions
- Files we used
- What we do next
```

This is exactly how Chapter 3 teaches users to begin — clean, structured, recoverable.

3.10 The Vizier’s Final Guidance on Beginnings

The Vizier told Amennu:

“If you start cleanly, the intelligence can follow you anywhere.
If you start vaguely, it will follow you everywhere.”

And then:

“Use one message to begin.
Use one message to checkpoint.
Use one message to restore.
This is all the discipline required.”

CHAPTER 4 — A Disciplined Workflow for Long-Running AI Projects

Most AI failures aren't caused by the model.
They're caused by a lack of structure.

This chapter gives you a practical, repeatable workflow that prevents drift, collapse, and rework—without requiring extra steps you won't actually follow. It is intentionally minimal: the smallest set of reliable habits that keep a long project alive.

The Vizier taught Amennu this workflow not because it is complicated, but because it is dependable.

4.1 The Three Golden Habits

Every successful long-form AI project rests on three simple behaviors:

1. **Checkpoints**
   Short, clear summaries of the current state.
2. **Artifact Bundles**
   All files needed to continue the work.
3. **Restoration Discipline**
   When a session degrades, you restart *with the checkpoint and files*, not by arguing with the model.

That's the entire system.

Everything else is optional refinement.

Vizier's Note:
"Order is not the enemy of creativity.
It is the ground on which creativity stands."

4.2 Habit One: Create Checkpoints at Natural Milestones

You do **not** need to checkpoint constantly.
Just at the right times:

- after finishing a section

- after completing an analysis pass
- after making major changes
- before starting a long chain of reasoning
- after incorporating new files

Each checkpoint should be:

- short (5–20 lines)
- factual
- neutral
- not prose
- not narrative
- not the whole conversation
- not instructions for the next step

Just **the current state**.

Example Checkpoint (Concise and Clean)

```
CHECKPOINT — Kitchen Remodel Project
Last Updated: 2025-01-14

GOALS:
• Expand kitchen by 4 ft.
• Add structural beam replacing load-bearing wall.
• Upgrade electrical to support induction range.

STATUS:
• Contractor A and B quotes collected.
• Floorplan v2 adopted as baseline.
• Permit application notes drafted.

OUTSTANDING DECISIONS:
• Confirm final cabinet layout.
• Select lighting plan (Option 2 or 3).

FILES INCLUDED (see uploads):
• floorplan_v2.png
• quotes_2025-01-12.pdf
• remodel_notes.txt
```

This takes less than a minute to produce, and the model can do 90 percent of it for you.

4.3 Habit Two: Keep Your Artifacts Organized and Versioned

Your files *are* the project.
The model only interprets what you give it.

Follow these minimal rules:

1. **Use descriptive names.**
   `floorplan_v2.png` is usable.
   `IMG_0473.png` is not.
2. **Use version numbers for drafts.**
   `electrical_plan_v3.pdf`
   `analysis_draft_v1.md`
3. **Keep an "Artifacts" folder for yourself.**
   So you can re-upload instantly.
4. **Upload only the files needed for the current task.**
   This reduces confusion and improves accuracy.

Vizier's Note:
"A scroll without a name is already half lost."

4.4 Habit Three: Restore Sessions Correctly

When drift appears—or when a session fails—you follow this exact sequence:

The Restoration Procedure

1. **Open a new chat.**
   Do not attempt to salvage the broken one.
2. **Paste your most recent checkpoint.**
3. **Upload all relevant artifacts.**
4. **Give this single instruction:**

```
Restore this project using the summary below, which is the latest checkpoint.
Treat the checkpoint as the authoritative record for this project.
Do not invent missing context or change prior decisions unless I explicitly
say so.

After reading it:
1) Briefly restate the current project state and plan.
2) List any files or artifacts you need me to re-upload.
3) Wait for my confirmation before doing new work.

---
[checkpoint content]
---
```

That’s all.

This avoids:

- residue from the broken session
- pattern contamination
- missing files
- missing instructions
- drifted goals
- wrong assumptions carried over

This method works because it re-creates the entire working memory cleanly.

4.5 How Often Should You Restore?

Restoration is not a last resort—it is maintenance.

You should restore when:

- drift symptoms appear
- the model contradicts earlier decisions
- a failure occurs
- the project becomes large or complex
- a new major task begins
- the session has been running for many hours

Every restoration **resets the statistical noise** that accumulates in transformers.

Think of it like clearing a whiteboard before starting a new step.

Vizier’s Note:
“Even the finest scribe grows tired. A fresh table restores his clarity.”

4.6 What NOT to Do

Beginners often sabotage themselves with well-intentioned behaviors.

Don’t:

- continue a session after drift has clearly begun
- argue with the model about what it forgot
- assume an upload persists
- assume the model knows what you meant
- rely on long, unstructured chats
- keep everything in one giant rolling thread
- attempt to “debug” a collapsed session

Do:

- checkpoint
- restore
- simplify
- continue cleanly

The entire point is to preserve precision while avoiding chaos.

4.8 The Philosophy Behind the Method

This workflow works because it matches the architecture:

- Checkpoints anchor meaning.
- Artifacts provide ground truth.
- Restoration resets attention patterns.
- Versioning preserves accuracy.
- Clean structure prevents drift.

It is not a hack.
It is not a trick.
It is the natural way to collaborate with a system that has:

- no persistent memory
- probabilistic reasoning
- finite working space

Vizier's Note:
"Order does not slow the work.
It keeps the work from vanishing."

CHAPTER 5 — The Vizier's Rules for Giving Instructions

Even with perfect checkpoints and disciplined restoration, a project can still drift if the instructions themselves are unclear, overloaded, or contradictory.

Large language models excel when the task is well-defined and fail when the user relies on implication, assumption, or conversational looseness.

The Vizier taught Amennu five rules for giving instructions that keep an LLM aligned across long projects. These rules are simple but transformative.

5.1 Rule One: One Task at a Time

LLMs do not juggle well.

When you combine multiple objectives in one message—*analyze this, rewrite that, generate new ideas, improve the previous section, and also check for errors*—the model must choose a dominant pattern. The result is inconsistency.

The fix is straightforward:

Separate tasks into steps.

Examples:

Bad:
"Rewrite this section, fix the mistakes, and also generate alternatives."

Good:

1. "Identify any issues in this section."
2. "Rewrite the section based on those issues."
3. "Generate 3 alternate variations."

The difference is clarity, and clarity is the source of stability.

Vizier's Note:
"A single hammer swing builds; a flurry of swings confuses the stone."

5.2 Rule Two: Use Explicit Constraints

LLMs weight constraints heavily when they are clear.

Strong constraints:

- “Use bullet points only.”
- “Keep the tone neutral and technical.”
- “Limit yourself to 5 lines.”
- “No metaphors.”
- “Use my definitions exactly.”

Weak constraints:

- “Be concise.”
- “Write better.”
- “Try to keep it short.”
- “Match my style.”
- “You know what I mean.”

If you want the model to obey a rule, name it cleanly.

Constraint Reinforcement

Restate constraints at natural transitions:

“Reminder: maintain R-3A-E tone. No narrative flourishes.”

This pulls constraints back into recency, preventing drift.

5.3 Rule Three: Provide Examples Whenever Possible

Transformers learn patterns *within the conversation.*
Your example becomes a template.

Examples act as:

- guardrails
- anchors
- style guides
- structure templates

Example: Good Prompting

```
Rewrite the following in the same structure as this example:

EXAMPLE:
• Clear claim
• Supporting detail
• Implication
• Next step

TEXT TO REWRITE:
[insert text]
```

The model will emulate the example more faithfully than any abstract description.

When Amennu supplied the Vizier with examples of contractor communications, the model's summaries and critiques immediately improved—because it now had patterns aligned to the domain.

Vizier's Note:
"A scribe who sees the form once will follow it forever."

5.4 Rule Four: Define Your Terms

Many failures come from undefined vocabulary:

- What counts as a "section"?
- What constitutes a "risk"?
- What is meant by "revise"?
- What is "concise"?
- What is "acceptable accuracy"?

LLMs infer definitions based on probability, not intent.

If you need precision, define your terms once and restate when needed.

Examples:

- "Concise = under 8 lines."
- "Technical tone = no analogies, no metaphors."
- "Checkpoint = 5–20 lines summarizing decisions, status, and pending tasks."
- "Draft v2 = incorporates changes from meeting notes only."

Define once → anchor → repeat when necessary.

5.5 Rule Five: Confirm Before Proceeding

Before letting an LLM carry a task forward—especially a long one—confirm understanding.

Simply ask:

"Repeat back the instructions in your own words."

Or:

"Summarize the constraints you will follow."

If the summary is wrong, the task would have been wrong.
If the summary is correct, the task will be stable.

This is the single most underused technique in advanced prompting.

Vizier's Note:
"A builder repeats the plan before laying the first stone."

5.6 How to Structure an Instruction Block

The Vizier recommended this template for complex tasks:

1. **Goal** — what you want
2. **Constraints** — rules the model must obey
3. **Inputs** — files, text, definitions
4. **Output Format** — what the answer should look like
5. **Examples** — patterns to emulate (optional but powerful)
6. **Confirmation** — "Repeat the instructions before beginning."

Example Instruction Block

```
GOAL:
Rewrite the executive summary for clarity.

CONSTRAINTS:
• Keep to 6–8 lines.
• Use formal technical tone.
• No metaphors or analogies.
• Preserve all factual content.

INPUT:
```

```
[insert text]

OUTPUT FORMAT:
A single 6–8 line paragraph.

BEFORE BEGINNING:
Repeat the instructions and confirm your understanding.
```

This structure eliminates ambiguity and prevents drift by front-loading clarity.

5.7 What Makes Instructions Fail

Even well-intentioned prompts can destabilize a model.
Here are the most common causes:

1. **Ambiguity**
   "Make it better" → better in what sense?
2. **Stacked tasks**
   "Rewrite, summarize, expand, and fact-check…" → competing intents.
3. **Conflicting constraints**
   "Be very concise, but also fully detailed."
4. **Lack of grounding**
   No examples, no definitions, no prior constraints.
5. **Overlong instructions**
   If the prompt is 1,000 words, the model will miss something.
6. **Implicit assumptions**
   The model does not know what you meant, only what you wrote.

Failures here lead directly to drift, confusion, and rework.

5.8 The Vizier's Instruction Checklist

Before sending a prompt, ask:

- Is it **one task**?
- Are the **constraints explicit**?
- Are my **definitions clear**?
- Do I provide an **example**?
- Did I specify the **output format**?
- Should I ask for **confirmation** before proceeding?

If the answer to all six is yes, the instruction will almost always succeed—even in long, complex, multi-file projects.

Vizier's Note:
"Clarity is kindness—to the scribe, and to yourself."

CHAPTER 6 — Advanced Checkpoint Strategies

Master Checkpoints, Branching, Ambiguity Resolution, and Quality Assurance

---

6.1 Why Advanced Checkpointing Matters

Basic checkpointing enables recovery.
Advanced checkpointing enables **coordination** — across tasks, across agents, and across time.

Large projects develop complexities such as:

- multiple concurrent workstreams
- parallel document branches
- integration points
- dependency chains
- ambiguous instructions
- multi-model workflows
- long restoration intervals

Advanced checkpointing ensures that a model:

- resumes work without drift,
- maintains alignment with project intent,
- isolates subtasks safely,
- and can be revived cleanly even after substantial temporal or contextual gaps.

Checkpointing does not eliminate ambiguity.
It **localizes** ambiguity and **limits** the cost of misalignment.

The remainder of this chapter formalizes how to use checkpoints at scale.

6.2 The Canonical Checkpoint Instruction (Hybrid Form)

The canonical request for a checkpoint is fixed and invariant.
Use it exactly as written:

```
"Request a checkpoint that includes:
- What we completed
- The current plan
- Important decisions
- Files we used
- What we do next."
```

This is the only instruction the user must memorize.

The model generates the checkpoint;
the user does not write them manually.

The structure is always the same.
The content reflects the current project state.

Vizier Commentary

The canonical instruction works because it imposes a strict structural schema.
The schema forces the model to:

- compress the session state into a usable summary,
- separate current decisions from future intent,
- identify dependencies and artifacts, and
- establish a clean continuation point.

Nothing about this structure changes.
All variation derives from *scoping*, not from rewriting the instruction.

Users must treat this instruction as a protocol command, not prose.

6.3 Master Checkpoints and Task-Specific Checkpoints

Large projects often require more structure than a single master checkpoint can provide. When a project is divided into subtasks or parallel efforts, a second checkpoint type provides this structure: the task-specific checkpoint. Master checkpoints capture the global project state; task-specific checkpoints define and initialize focused lines of work. Work Logs, which are not checkpoints, record what happened during subtasks without modifying the project's restoration point.

6.3.1 Master Checkpoints

A master checkpoint is created using the canonical instruction with **no scoping qualifier**:

```
Request a checkpoint that includes:
- What we completed
- The current plan
- Important decisions
- Files we used
- What we do next
```

Master checkpoints:

- capture the entire project state,
- act as lineage anchors,
- preserve global decisions,
- and provide universal restoration points.

They are used at:

- major milestones,
- structural transitions,
- handoffs across sessions or models,
- and before branching into subtasks.

6.3.2 Task-Specific Checkpoints

Task-specific checkpoints are used to launch a sharply defined subtask inside a larger project. They create a clean working context that isolates the subtask from unrelated project details. They do not modify or update a master checkpoint.

The core of a task-specific checkpoint is the **task instruction**. This acts as the subtask's Opening Phrase and should follow the same principles described in Section 3 (Opening Phrases). As with initial prompts, the wording here determines the trajectory and quality of the subtask session.

A task-specific checkpoint includes:

- A clearly written **task instruction** (the subtask's Opening Phrase)
- The minimal background context required
- The specific goal or deliverable for the subtask
- Any constraints, definitions, or reference materials
- Success criteria for the subtask
- Any known risks or decision points inside the subtask

When the subtask is complete, it returns to the master thread with either:

- A completed deliverable (ideal), optionally accompanied by a Work Log, or
- A Work Log indicating the subtask is complete

The master thread reviews these outputs and continues the project. No additional checkpoint type is involved.

For guidance on writing strong task instructions, see Section 3 on Opening Phrases, which explains how task framing determines the entire reasoning trajectory.

How to Request a Task-Specific Checkpoint

Ask the LLM to generate a task specific checkpoint using this template and your structured task instruction.

```
Create a task-specific checkpoint for a new subtask.

Include:

- A structured task instruction (the Opening Phrase for this subtask)
- The minimal context this subtask needs
- What the subtask must accomplish or produce
- Any constraints, definitions, or reference materials needed
- Success criteria for the subtask
- Any risks, ambiguities, or decisions expected during the task
```

This task-specific checkpoint launches a separate subtask and does not modify the master checkpoint.

6.3.3 Returning Subtask Results to the Master Thread

When a subtask launched by a task-specific checkpoint is complete, its results must be returned to the master thread in a clear, usable form. There are two successful patterns:

1. **A completed deliverable** (ideal), optionally accompanied by a Work Log.
2. **A Work Log that indicates the subtask is complete**, including where any deliverable can be found if stored externally.

A Work Log is a concise summary of what the subtask accomplished: what was done, what changed, what was produced, and any suggestions for how the main project should proceed. It is not a checkpoint and does not store project state.

To continue the main project, copy the deliverable and/or Work Log into the master thread. The master session incorporates the information as part of its normal workflow.

Generic Work Log Template

Paste the following into the model when you want a structured Work Log:

```
Create a Work Log that summarizes the completed work in this subtask.

Include:

- What was done in this subtask
- What was produced (the deliverable), or where it can be found
- Key decisions made
- Any changes to assumptions, definitions, or understanding
- Any risks, dependencies, or open questions
- Recommendations for how the main project should proceed

Write the Work Log so it can be copied into the master thread and
understood without referencing the rest of this conversation.

Do not generate a checkpoint. This is a Work Log only.
```

6.4 Ambiguity Handling: When the Model Introduces Uncertainty

Checkpoints preserve structural state, not semantic intent.
As a result, ambiguity may arise even when user instructions are sound.

High-quality models frequently detect this and pause rather than corrupt prior work.

Two examples illustrate the pattern.

6.4.1 Example 1 — Model-Introduced Ambiguity

A user prepares to resume work after creating a task checkpoint.

User:
“Request a checkpoint for beginning the next session focused on refining the client guide, using the standard structure.”

The model generates the checkpoint.

In the next session, the user follows phrasing the model previously suggested:

User:
“Proceed with the merged client guide with all improvements integrated.”

The model halts and returns:

Model:
“There are two valid interpretations:
A) Apply the improvements to the existing guide only
B) Merge the client guide with the technical reference to create a unified volume
Which should I follow?”

The ambiguity originated in the *model’s internal interpretation*, not in the user’s intent.

Resolution

User:
“Apply the improvements to the existing guide only.”

The model immediately realigns and executes correctly.

Principles Demonstrated

- Checkpoints do not encode semantic choice.
- The model may surface multiple valid continuations.
- Pausing is protective behavior.
- A short correction fully reanchors the session.

6.4.2 Example 2 — Scope Ambiguity

A user begins work from a task checkpoint.

User:
“Integrate these notes into the document.”

The checkpoint contains multiple sections from prior work.
The model evaluates:

- Should the notes be integrated into the executive summary?
- The background section?
- The methodology section?
- A pending appendix?

Model:
“To integrate the notes, I need to know which section you intend.
Please specify the target location.”

Resolution

User:
“Integrate them into the background section only.”

The model applies the instruction correctly.

Principles Demonstrated

- Checkpoints preserve structure, not scope.
- Vague target objects (“the document”) require user clarification.
- Minimal correction is sufficient.
- The user supplies intent; the model supplies execution.

6.5 Checkpoint Quality Verification

Checkpoints can occasionally be incomplete — e.g., truncated output or missing sections. Verification should be fast and mechanical.

A valid checkpoint must contain:

1. **What we completed**
2. **The current plan**
3. **Important decisions**
4. **Files we used**
5. **What we do next**

If any component is missing or the text ends mid-sentence, respond:

“This appears incomplete.
Regenerate the checkpoint using the full structure.”

The model will correct automatically.

Checkpoint verification is a lightweight discipline, but essential for preventing drift.

6.6 Branching Strategies for Multi-Deliverable Work

Task-specific checkpoints enable safe branching for complex projects such as:

- multi-document deliverables,
- research workflows,
- software development cycles,
- creative production pipelines,
- and analytical reporting.

A branching pattern typically follows:

1. Create a master checkpoint.
2. Create task checkpoints for each deliverable.
3. Work on each branch independently.
4. Merge results manually or with model assistance, using deliverables and Work Logs as needed.
5. Once the merged content is stable, request a **new master checkpoint** to capture the updated project state.

This prevents task interference and ensures each branch remains coherent.

6.7 Real-World Application Examples

Example: Documentation Suite

- Master checkpoint anchors the suite
- Task checkpoints for:
  - API overview
  - Quick-start rewrite
  - Troubleshooting section
- Each branch proceeds independently
- A final review merges the outputs, followed by a new master checkpoint

---

Example: Research Project

- Master checkpoint stores the overall study design
- Task checkpoints for:
  - literature review
  - dataset analysis
  - interpretation drafting
- Each workstream avoids polluting the others

Example: Marketing Campaign

- Master checkpoint captures messaging framework
- Task checkpoints for:
  - landing page copy
  - email sequence
  - ad variations
- Final session merges content into a cohesive campaign

Example: Product Development

- Master checkpoint binds specifications
- Task checkpoints for:
  - requirements
  - implementation notes
  - test plan
  - release notes
- Cross-team work is isolated but consistent

6.8 The Vizier's Advanced Principles for Checkpointing

1. **The user never writes checkpoints.**
   The user requests them.
2. **All checkpoints follow the same five-part structure.**
   The model generates the content.
3. **Master checkpoints anchor the project.**
   Task checkpoints isolate subtasks.
4. **Checkpoints preserve state.**
   Users preserve intent.
5. **Ambiguity is expected.**
   Correction must be minimal and authoritative.
6. **A checkpoint is a control surface:**
   it restores discipline across time and across models.
7. **Verification is mandatory.**
   Completeness is non-negotiable.
8. **Advanced workflows depend on branching.**
   Each branch begins with a task-specific checkpoint.

With these principles, users can execute extended, multi-agent, multi-session work reliably, with high structural fidelity and minimal drift.

CHAPTER 7 — When You're Ready to Go Further

By the time you master checkpoints, artifact control, drift detection, and recovery, you already possess the single most important skill in modern AI work:

You can keep a project coherent over time.

Everything that follows in this chapter builds on that foundation.
These are the areas users naturally grow into once they stop using AI for one-off tasks and begin using it for *workflows*, *systems*, and *automation*.

The details in this chapter are intentionally medium in depth: enough to show the landscape and prepare you for what comes next, without locking in techniques that continue to evolve rapidly. Future editions of this guide will expand on many of these areas.

The Vizier's Note:
"Master your scrolls and your checkpoints first. When the foundation is solid, higher work becomes simple."

7.1 From Interactive Answers to Structured Workflows

Most users begin by asking an intelligence isolated questions.
Eventually, they discover something more powerful:

The model can assist in multi-step, multi-day, multi-asset work.

This transition looks like:

- moving from ad hoc prompts
- to structured sequences
- to repeatable patterns
- to workflows you use again and again

The critical shift is not in the model — it is in *you*.
The moment you start treating an LLM not as a "genie," but as a *component in a larger process*, your capabilities expand dramatically.

This is where Chapter 3's discipline becomes the backbone for everything else.

7.2 Designing Repeatable AI Workflows

Once you complete a long project successfully, you will notice something important:

You are repeating the same patterns in every new project.

Examples:

- a consistent opening message
- a consistent checkpoint layout
- a consistent artifact structure
- a consistent way of restoring a session

This is where repeatability begins.

A repeatable workflow typically includes:

1. **A standardized project start template**
   ("Here's the project, the assets, what I need, and your role.")
2. **Task-specific prompt templates**
   ("Rewrite this section for clarity using rules A, B, C.")
3. **A consistent checkpoint cadence**
   (every major decision, every deliverable, every upload cluster)
4. **A clean artifact hierarchy**
   (v1, v2, v3 folders; input, output, reference; dates or versions)
5. **A structured recovery flow**
   (always the same sequence of restore → confirm → resume)

You will eventually discover that these workflows *are portable* — you can apply them in code, in APIs, in batch-processing, or across multiple models.

The Vizier's Note:
"A disciplined process becomes a path others may follow."

7.3 Introducing Tool Use and Automation

Once a workflow stabilizes, a natural question arises:

Can I automate the repetitive parts?

This is the moment many users begin using AI to write small tools that support the workflow:

- Python scripts to sort or rename files
- Shell scripts to bundle artifacts
- Code to extract text from PDFs
- Scripts that apply the same transformation to many files
- API calls that automate a repetitive LLM task

These are not “agents.” They are simply tools — extensions of your workflow.

A typical path looks like:

1. **Manual process becomes repeatable workflow**
2. **Workflow becomes semi-automated via scripts**
3. **Scripts become AI-generated or AI-assisted**
4. **The AI begins to manage parts of the workflow itself**

This is also where you begin designing **validation steps**, because automation without verification simply automates error.

Future editions of this guide will expand heavily in this direction — Python pipelines, batch processing, artifact-aware automation, and API-driven reasoning loops.

7.4 Working Across Multiple Models

Different models specialize in different strengths:

- ChatGPT: structured reasoning, code generation, system reliability
- Claude: long-form analysis, detail coherence, narrative refinement
- Gemini: multimodal merging, visual + textual integration

Eventually, you will discover the real frontier:

Combining models into a coordinated workflow.

Example:

- Claude produces the detailed outline
- ChatGPT generates the code and tests it
- Gemini extracts structure from the images or PDFs
- You integrate everything with consistent checkpoints

When you do this, something subtle but profound happens:

Your checkpoint and artifact discipline becomes the *interface* between models.

If you can keep a single LLM coherent, you can keep multiple coordinated.

This area will grow rapidly as models become more specialized.

7.5 Scaling to Large, Multi-Artifact, Multi-Phase Projects

In serious work, you are not managing one file — you are managing dozens or hundreds:

- documents
- transcripts
- datasets
- diagrams
- images
- revisions
- versions
- notes
- decision logs

This is where the discipline from Chapters 3 and 4 becomes essential.

Key principles when scaling up:

1. **Version your artifacts clearly**
   If you have 20 drafts, the model must always know which one is current.
2. **Segment the project into phases**
   When a session becomes too large, you end it intentionally and start fresh.
3. **Maintain a master index**
   A human-readable list of files, versions, and roles.
4. **Use the model to help manage the hierarchy**
   The LLM can maintain the *map* as long as you maintain the *ground truth*.
5. **Never let ambiguity accumulate**
   Drift in file versions is one of the fastest ways to corrupt complex projects.

The Vizier's Note:
"When the archives grow large, the labels save you."

7.6 Reliability Principles for Advanced Workflows

Once your work grows more ambitious, reliability stops being optional.

There are four emerging principles for dependable advanced use:

1. Verification Before Integration

Never trust a model's long output blindly.
Require the model to:

- check its work
- show its assumptions
- compare versions
- identify inconsistencies

2. Redundancy Through Cross-Model Checks

If the work is high-stakes, run the same task through two different models and compare.

3. Frequent Re-Anchoring

The more complex the project, the more often you should restore from a known-good checkpoint.

4. Clear Separation Between Work and Summary

Mixing them accelerates drift.
Keep summaries distinct and regenerate them often.

These patterns are still evolving, but they will not vanish — automation increases the cost of error, and thus increases the value of discipline.

7.7 Preparing for AI Agents (the Real Kind)

True agents — not hype, not marketing — are simply workflows that run *without a human in every loop*.

But agents still require:

- clear steps
- stable intermediate states
- strong guardrails
- explicit memory interfaces
- verification logic
- recovery paths

In other words:

Agents are checkpointed workflows with code around them.

Everything you learned in this guide becomes the agent's backbone.

This field will expand dramatically over the next 1–3 years.
Future editions will explore:

- task graphs
- long-horizon planners
- tool-enabled reasoning
- agent reliability frameworks
- mixed-initiative human workflows

But you are already farther along this road than most — you have the fundamentals.

The Vizier's Note:
"An agent is merely a scribe who knows which task comes next."

7.8 What This Guide Cannot Yet Teach

AI capabilities shift every quarter.
Entire classes of workflows—video pipelines, multimodal reasoning, agent orchestration—are changing too quickly for stable long-form instruction.

This guide therefore avoids:

- brittle platform specifics
- short-lived API patterns
- ephemeral UI details
- tricks that depend on undocumented behavior
- workflows that are likely to be obsolete in six months

Instead, it teaches the *durable layer*:

- verification
- preservation
- context control
- restoration
- detection
- correction
- disciplined structure

The surface changes.
The fundamentals persist.

7.9 Closing Words from the Vizier

“When you began, you asked only for answers.
Now you understand the work:
to preserve what matters,
to see drift before it grows,
to restore what is lost,
and to build with intention.

All intelligences change.
All tools evolve.
But discipline ages slowly.

The future will bring new systems,
new capabilities,
new forms of intelligence.

Do not chase them.
Build the habits that let you use any of them well.

You now possess those habits.
Walk forward with confidence.
Your next project will be easier than your last.”

ABOUT THIS GUIDE

What Is the Vizier's Guide?

The Vizier's Guide to Managing Artificial Intelligences is a practical manual for using AI systems on long-running, complex projects. It focuses on fundamentals that matter across every major platform — ChatGPT, Claude, Gemini — because it teaches **principles**, not platform tricks.
Context discipline, checkpointing, artifact management, recovery workflows, drift detection, and multi-session continuity are universal challenges.
This guide offers tested solutions.

The guide is **human-authored with AI assistance**. It was built collaboratively: some paragraphs began with the human author and were refined by an AI; others began with an AI and were reshaped by the human. The boundary between those contributions is intentionally blurred. What matters is not who wrote which sentence, but whether the result is useful, repeatable, and reliable.

The scenarios described throughout — crashes, missing content, silent degradation, context drift, corrupted sessions — are not hypotheticals. They occurred in real work.
A prior edition of this guide experienced a session failure in which several newly generated sections were never captured in the artifact bundle. Reconstruction was possible only because an earlier session had been preserved. This real incident directly informed the checkpoint and artifact practices recommended in this edition.

This version is part of an evolving document lineage.
Earlier editions were created during the era of strict token limits, when losing context after a few thousand tokens was normal. Practices like checkpoints and recovery points were born of necessity. Modern systems have vastly larger windows, but the underlying risks — silent drift, overwritten assumptions, vanishing uploads, missing artifacts — remain.
This rewrite reflects the state of LLM capabilities as of late 2024 and the techniques that remain essential even as platforms evolve.

AI systems change quickly. Token limits expand, new models appear, capabilities shift, and user interfaces are redesigned. Some specifics in this guide may be out of date within six months.
The **principles**, however — verify, preserve, monitor, reconstruct — should remain relevant far longer.
The commands may change.
The discipline will not.

Future Directions

Future editions will explore advanced workflows, especially **AI-assisted automation**.
When you use an AI model to generate Python scripts that process your data, orchestrate multi-step tasks, or run analyses in batch, you are already building agents.
When you extract reusable routines from repeated interactions, you are formalizing an AI-powered workflow.
These are emerging practices; as the author gains more experience turning interactive reasoning into automated systems, new chapters will follow.

Areas of planned expansion include:

- Multi-agent orchestration
- AI-generated code pipelines
- Visual media workflows
- Video generation and transcription (SORA and successors)
- Multimodal artifact integration
- Long-horizon project automation
- Reliability frameworks for autonomous tools
- Methods for detecting hidden degradation in long chains of reasoning
- AI-driven consistency checks for large documents and datasets

Not every reader will move into these areas immediately.
Many will first use AI for creative projects — images, videos, narrative work — before applying it to professional workflows. That's natural.
This guide is written for readers who are ready to use AI **not as a novelty**, but as a working tool that supports meaningful, high-stakes, multi-phase work.

This guide will continue to evolve.
The latest version will always be available through the author's Discord account, free to read and free to share.
Feedback, corrections, and real-world examples are welcome.
The field moves quickly, and practical knowledge grows faster when it is shared.

Copyright © 2024 *Dunker Fosen*. All rights reserved.

When sharing or referencing this guide, please credit **@dunkerfosen on Substack**.

Appendix A — Operational Templates

COMPANION TOOL

The **Vizier Preservation Toolkit** automates checkpoint generation, artifact tracking, and session restoration. Recommended for easier implementation of this methodology. Free and open-source. No registration, no paywalls, no gatekeeping.

Available at: https://vizier-labs.github.io/vizier-preservation-toolkit/

This appendix is not a stack of forms for you to fill in by hand.
It is a set of **canonical prompts and interaction patterns** you can reuse with any modern LLM.

Each template follows the same pattern:

1. **When to use it**
2. **What you say to the model**
3. **What you should expect back**
4. **What you do with the result**

The intelligence does the work.
You preserve the work.

A.1 How To Use These Templates

Use these templates exactly as written the first few times.

Over time, you will adapt them to your own projects.
When you do, keep three rules in mind:

1. The model generates the structure.
2. You sanity check and correct.
3. You save the output as a checkpoint or artifact.

If a template ever feels like “homework for the human,” you are doing too much.

A.2 Starting A Project

When to use:
At the beginning of any substantive project that will last more than one or two turns.

Prompt to the model:

```
I am beginning a new project and I want your help managing it.

PROJECT:
[Describe the project clearly in 1–2 lines.]

ASSETS I HAVE:
[List documents, data, images, notes, drafts, or other artifacts.]

YOUR ROLE:
[Define exactly what I should use you for. Examples: track decisions,
summarize work, analyze data, rewrite sections, maintain coherence.]

TIMELINE:
[If this will last more than one session, say so.]

First, restate the project in your own words, propose a simple structure for
how we should work together, and list what you need from me to start.
```

What to expect back:
A concise restatement, a simple plan, and a request for specific files or information.

What you do:

- Fix anything the model misunderstood.
- Provide the requested inputs.
- Then move to checkpoints as soon as you have a real milestone.

A.3 Requesting a Checkpoint

When to use:
Whenever you hit a meaningful milestone or before a risky or long sequence of steps.

Prompt to the model:

```
Create a checkpoint for this project that includes:

- What we completed
- The current plan
- Important decisions
- Files we used
- What we do next

Keep it short and structured so I can save and reuse it later.
```

What to expect back:
A compact, structured block of text. Not a transcript, not a narrative, just state.

What you do:

1. Scan for errors or omissions.
2. If needed, say:
3. `Update the checkpoint to fix the following issues:`
4. `[list corrections briefly]`
5. Copy the final checkpoint into your own storage with a clear name, for example:

   ```
   ProjectName_Checkpoint_2025-12-04.txt
   ```

You never build the checkpoint by hand.
You only ask for it, correct it, and save it.

A.4 Restoring From Failure Or From A Break

When to use:
After a crash, obvious drift, UI reset, or when you are returning to the project in a new chat.

Your steps with the model:

1. Open a new chat.
2. Paste your most recent checkpoint.
3. Re-upload the relevant artifacts.

4. Then issue this prompt:

```
Restore this project using the summary below, which is the latest checkpoint.
Treat the checkpoint as the authoritative record for this project.
Do not invent missing context or change prior decisions unless I explicitly
say so.

After reading it:
1) Briefly restate the current project state and plan.
2) List any files or artifacts you need me to re-upload.
3) Wait for my confirmation before doing new work.

---
[checkpoint content]
---
```

What to expect back:
A clean reconstruction of state based only on the checkpoint and files.

What you do:

- Confirm it matches your understanding.
- If it does not, correct it explicitly, or paste an earlier checkpoint and repeat.

A.5 Drift Check

When to use:
When you suspect things are starting to go off, but the session has not fully collapsed.

Prompt to the model:

```
Perform a drift check on this project.

1. Compare the current working state against the last checkpoint.
2. Identify any inconsistencies in facts, decisions, goals, or constraints.
3. Flag anything that appears to have changed meaning without being
explicitly updated.
4. Report the issues clearly, without trying to fix them yet.
```

What to expect back:
A list of possible drifts: changed numbers, missing constraints, altered definitions, conflicts.

What you do:

- Decide if the drift is minor or major.
- For minor drift, correct and continue.
- For major drift, stop and perform a full restore from the last known good checkpoint.

A.6 Instruction Block Wrapper

When to use:
Whenever you give a complex or high impact task.

Prompt to the model:

```
I am going to give you a complex task. Use this instruction block and confirm
before you begin.

GOAL:
[What the task is.]

CONSTRAINTS:
- [Tone, length, format, rules]
- [...]

INPUTS:
- [Text, data, files, definitions you will use]

OUTPUT FORMAT:
[Exactly what the answer should look like.]

Before you begin, restate the instructions in your own words and list the
constraints you will follow.
Wait for me to confirm or correct your understanding.
```

What to expect back:
A restatement of the task and constraints.

What you do:

- If the restatement is correct, say:
- `Confirmed. Proceed.`
- If not, correct and ask the model to restate again.

This is cheap insurance against misaligned work.

A.7 Correction And Realignment

When to use:
When the model states something wrong that matters for future steps.

Prompt to the model:

```
Correction:

[State the correct fact, rule, or decision in one or two sentences.]

Update the working summary or checkpoint so that:

- The incorrect information is removed or fixed.
- The correct version is clearly stated.
- Future work will follow the corrected version.

Show me the updated summary or checkpoint.
```

What you do:
Save the updated summary or checkpoint if it is correct.

A.8 Using The Model To Help With File Names And Structure

The book gives naming rules because they matter, but you do not have to invent every name yourself. You can have the model propose them, then you adopt and standardize.

When to use:
When you have multiple artifacts and want a clean naming scheme.

Prompt to the model:

```
We need a clear, consistent naming and folder structure for this project's
artifacts.

1. Propose a short, human readable naming convention for files that includes:
   - Project name or code
   - Item type (draft, data, notes, summary, checkpoint)
   - Version or date

2. Propose a simple folder structure for:
   - Inputs
   - Outputs
   - Checkpoints
   - Archives or old versions

Keep the proposal simple enough that I will actually follow it.
```

What to expect back:
A concrete pattern, for example:

- `Project_Draft_v2.docx`
- `Project_Data_2025-02-14.csv`
- `Project_Checkpoint_2025-02-14.txt`

What you do:

- Accept or tweak the scheme.
- Then stick to it and treat it as part of the project contract going forward.

A.9 Task Transition Template

When to use:
When switching from one major task to another inside the same project.

Prompt to the model:

```
We are switching tasks in this project.

1. Briefly summarize our current state in 3–5 lines.
2. Close out the prior task clearly.
3. Restate the new task:

[Describe the new task.]

4. List any constraints from earlier that still apply.

Wait for my confirmation before starting the new task.
```

What you do:

- Confirm the transition and constraints.
- Or correct them before allowing the model to proceed.

Appendix B — Glossary

This glossary defines terms as they are used in this Guide.
They are operational definitions: each reflects a specific behavior, risk, or discipline relevant to managing modern LLMs.

The purpose is not academic completeness.
It is practical clarity.

Use these definitions when issuing instructions, evaluating outputs, training colleagues, and designing workflows.

Artifact

Any file, draft, summary, or structured output produced during the project.
Artifacts are the external memory the model does *not* have and cannot manage on its own.

Checkpoint

A state snapshot generated *by the model* that includes:

- what has been completed
- key decisions
- the current plan
- relevant files
- what comes next

You request it, correct it, and save it.
It is the backbone of multi-session work.

Checkpoint Drift

When the model's current working state starts to diverge from the information recorded in the checkpoint.
Early symptoms include: changed numbers, missing constraints, reinterpreted goals, or unexplained shifts in meaning.

Collapse

A session failure where the model:

- contradicts previous decisions,
- loses track of constraints,
- improvises structure,
- or forgets earlier context.

A collapse requires restoring from a checkpoint, not trying to "coach" the failing session back.

Compression

A side effect of long, dense conversation in which earlier details are summarized, blurred, or overwritten inside the model's internal context representation.
Compression is *not* the same as forgetting; it is distortion.

Context Window

The finite working memory of a model.
Everything the model "knows" in a session must fit inside it.
Overflow leads to compression and drift.

Constraint

A rule the model must follow for a specific task:
tone, length, format, scope, or method.
Constraints stabilize output and prevent improvisation.

Decision

A user-approved choice that becomes part of project state.
Decisions belong in checkpoints and must be preserved across sessions.

Drift

Gradual deviation from constraints, definitions, decisions, or tone.
Drift begins subtly and, if uncorrected, results in collapse.

Drift Check

A structured prompt requesting the model to:

- identify inconsistencies
- compare against the last checkpoint
- flag contradictions
- report state deviations

A drift check is diagnostic, not corrective.

Explicit State

All information the model must use that is not already in context:
checkpoints, files, decisions, summaries.
The AI does not maintain state on its own; the user supplies it.

Failure Mode

A predictable way an LLM session degrades: drift, contradiction, collapse, hallucination, or unanchored improvisation.
Failure modes are structural, not emotional; they are expected behaviors.

Hallucination Pressure

When a model produces content that fits a pattern even when no factual basis is available.
This is not deception; it is pattern continuation without grounding.

Instruction Block

A structured wrapper for complex tasks that includes:

- goal
- constraints
- inputs
- output format
- optional example

It reduces misinterpretation and enhances repeatability.

Pattern Continuation

The model's default behavior: extend the most likely pattern from the current input.
All other behaviors flow from this principle.

Recency Bias

The tendency of a model to overweight the most recent turns and underweight earlier details, especially under compression or long-session load.

Restoration

Reconstructing project state in a new session using:

- a checkpoint
- the necessary artifacts
- a restoration prompt

Restoration resets the project without carrying contamination from prior drift.

Role Alignment

The clear separation of responsibilities:

- the model generates structure, summaries, checkpoints, drafts
- the human maintains constraints, reviews output, preserves state
  Misalignment causes drift and collapse.

State

The combination of:

- current goals
- confirmed decisions
- project plan
- artifact set
- constraints

State must be explicit; the model does not infer it reliably.

Structure

The visible scaffolding of a task: headings, sections, format, numbering, bullet logic.
Structure must be *given* or *requested*; if left to the model's improvisation, drift increases.

Task Transition

A structured prompt used when switching tasks.
It closes the previous task, defines the new one, and prevents pattern bleed-through.

Versioning

A system for naming artifacts so they remain trackable over time.
Good versioning prevents accidental reuse of outdated files.

Vizier's Principle

The guiding doctrine of this Guide:
You are the memory.
The model does not carry context across time unless you supply it, preserve it, and restore it.

COPYRIGHT AND LICENSE SECTION

For *The Vizier's Guide to Managing Artificial Intelligences*
(Full Guide Version)

Copyright © 2024 Dunker Fosen. All rights reserved.
First Edition December 2024
Canonical edition: PDF archived at the Internet Archive.

This work includes narrative content, operational methodology, instructional frameworks, procedural checklists, structural patterns, and terminology created by the author. All components are protected by copyright, trademark, and applicable intellectual property law.

LICENSE

This work is licensed under Creative Commons Attribution NonCommercial ShareAlike 4.0 International (CC BY NC SA 4.0).
Full license: https://creativecommons.org/licenses/by-nc-sa/4.0/

You may:

- Share the work in any medium
- Remix, adapt, analyze, and build upon it
- Use the methodology for personal, educational, and research purposes
- Develop noncommercial derivative guides, references, or tools
- Teach the concepts in noncommercial academic or community settings

Conditions:

- Attribution: Credit "Dunker Fosen" and link to the canonical source
- NonCommercial: No commercial use of any part of this work
- ShareAlike: Derivatives must use the same CC BY NC SA 4.0 license
- Integrity: Represent derivative changes accurately without implying endorsement

---

NONCOMMERCIAL DEFINITION

To avoid ambiguity, "noncommercial" excludes:

• Paid trainings, workshops, webinars, bootcamps
• Consulting engagements, coaching, or professional services
• SaaS platforms, automation frameworks, or AI tools using this methodology
• Subscription newsletters, private communities, or monetized publications
• Commercial software libraries, prompt packs, or model templates
• Internal corporate tooling or enterprise implementations
• Summaries or adaptations sold for profit
• Inclusion in any product or service exchanged for monetary value or business benefit

If it generates revenue or business value, it requires a commercial license.

PROHIBITED USES WITHOUT WRITTEN PERMISSION

Commercial licensing is required for:

• Incorporating methods or patterns into software products or tools
• Building agents, copilots, or AI systems using this structure
• Creating paid courses, tutorials, or certification programs
• Publishing derivative guides or frameworks for profit
• Bundling any portion of this guide into commercial training products
• Using the Vizier terminology, narrative, branding, or methodology in commercial offerings
• Creating SaaS or enterprise automation systems based on this work
• Using this text as part of datasets for training commercial AI models
• Creating commercial prompt libraries or LLM fine-tuning corpora from this work
• Using any chapter structure, sequence, or instructional framing commercially

Commercial licensing inquiries: fosendunker@gmail.com

AI TRAINING AND MODEL USAGE RESTRICTIONS

This work may not be:

• Used as training data for commercial AI models
• Incorporated into fine tuning datasets or embedding corpora
• Reproduced in prompt libraries or commercial automation systems
• Scraped or mined for model development without permission
• Used to generate commercial derivative content through automated means
• Integrated into commercial LLM agents, copilots, or frameworks

Academic research use is permitted with attribution.

TRADEMARK NOTICE

“The Vizier’s Guide”, “Vizier Methodology”, “Vizier’s Beginner Guide”, and “Vizier” as applied to AI project management frameworks are trademarks of Dunker Fosen.

These marks may not be used in:

• Branding or naming software, tools, methods, or frameworks
• Advertising or marketing of commercial services
• Titles of courses, certifications, books, or training materials
• Any commercial offering implying affiliation or endorsement

Permission required for all commercial use.

CANONICAL EDITION AND SOURCE OF TRUTH

The canonical and authoritative version of this work is the PDF archived in the Internet Archive. Other versions, including Substack reproductions or excerpts, may differ in formatting.

Always cite the canonical edition.

HOW TO CITE THIS WORK

Academic or professional:
Fosen, D. (2024). *The Vizier’s Guide to Managing Artificial Intelligences*.
Canonical edition retrieved from the Internet Archive.

In-text citation: (Fosen, 2024)

Blog, article, or online media:
“Based on Vizier methodology by Dunker Fosen”
Link: https://substack.com/@dunkerfosen

Software or tools:
“Checkpoint methodology adapted from The Vizier’s Guide by Dunker Fosen”

DERIVATIVE WORKS (NONCOMMERCIAL)

If you create noncommercial derivatives, you must:

1. Attribute the original work to the author
2. License the derivative under CC BY NC SA 4.0
3. Link to the canonical edition
4. Identify modifications clearly
5. Avoid implying endorsement or collaboration

ENFORCEMENT

Unauthorized commercial use or trademark misuse may result in:

• DMCA takedowns
• Cease and desist demands
• Removal from commercial platforms
• Copyright infringement claims
• Claims for damages and attorney fees
• Enforcement under trademark law

CONTACT

General inquiries: fosendunker@gmail.com
Commercial licensing: fosendunker@gmail.com
Report violations: fosendunker@gmail.com
Substack: @dunkerfosen